RasPi

DESIGN
BUILD
CODE

41

Get hands-on with your Raspberry Pi

REVEALED! THE LIGHTEST PI TABLET EVER

MAKE A PI-POWERED XMAS JUMPER!

Plus Hack a friend's Minecraft World

Welcome

Listen closely and you can hear the not-so-distant sound of sleigh bells as the Christmas season draws ever closer, bringing with it associated office revelries like parties, secret santa and the increasingly popular Christmas Jumper Day.

For the past few years employees have been locking horns in a one-day battle of yuletide one-upmanship involving the finest festive knitwear a network administrator can muster. This year, Raspi magazine is on hand to tip the balance in your favour, with a tutorial on making a pi-powered pull-over that will surely conquer all.

There are some less seasonal projects for the grinches among you too, including a look at the smallest Pi tablet in existence and a tutorial on hacking a friend's minecraft world.

Get inspired

Discover the RasPi community's best projects

Expert advice

Got a question? Get in touch and we'll give you a hand

Easy-to-follow guides

Learn to make and code gadgets with Raspberry Pi

Editor

From the makers of

LinuxUser & Developer

Join the conversation at...

@linuxusermag | Linux User & Developer | linuxuser@futurenet.com

Contents

Get into the seasonal swing: Build a social media sweater

Augment the traditional cheesy festive jumper with LEDs that light up every time you receive a tweet

It is fast approaching the winter holidays. Decorations, egg-nog, carols, Home Alone and, of course, Christmas is not Christmas unless you don't celebrate Christmas of course, without a festive jumper (or sweater to our US readers). The gaudier the better and extra points if it is knitted. So, how about augmenting your jumper with LEDs so that while you make small talk at the office party or throw some shapes on the dance floor, your fellow colleagues can trigger the LEDs via social media? Each time they do, your jumper will light up with all the joy of the season and send them a confirmation tweet with suitable festive image to thank them.

In this tutorial, we'll take some Christmas lights or old LEDs and write a Python program which checks your Twitter timeline for a specific 'trigger' word. The Pi Zero W is the perfect size for embedding into the jumper and the built in Wi-Fi means you can tether it to your mobile device, meaning that anyone can help you light up and spread the Christmas cheer this holiday. You can see it in action here: **https://youtu.be/5pcouPXzPEM**.

THE PROJECT ESSENTIALS

Pi Zero

USB portable power supply

Suitably gaudy winter sweater

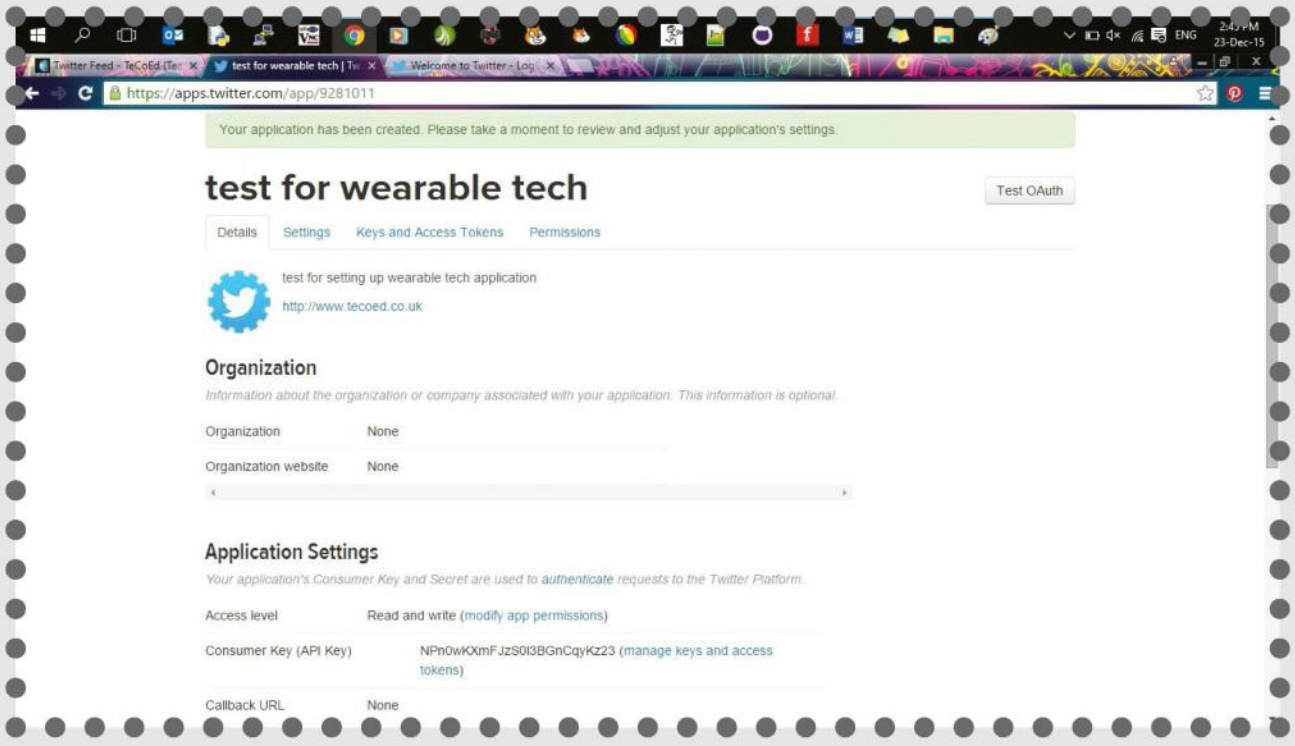
Battery-powered LED Christmas lights

01 Sign up for Twitter API keys

To interact with Twitter, you will require a set of consumer and authorisation keys, which are available from the Twitter API site. If you already have these keys, you can jump to **Step 3**. If not, then head on over to **https://apps.twitter.com** and sign in with your regular Twitter username and password. Select the 'create a new app' button. Fill in the details for your app as required, then tick and confirm that you accept the terms and conditions. On completion, you will be taken to the app overview page. Here you will need to select the 'Access Level Permission' for your App. For this project, you will need to set the permission at 'Read and Write'.

02 Set up the Wi-Fi

Now set up your 'keys'. These are the codes which are used to access your Twitter account via Python. Click the 'keys and access token' page. You will be presented with your consumer key and consumer secret; copy these down as they are required in **Step 7**.



At the bottom of the page you will find the 'access tokens'. Simply press the button and this will generate them. Again, note these down for use in **Step 8**. Note that each time you press the button, a new set of keys are created; this is useful if you think they have become compromised.

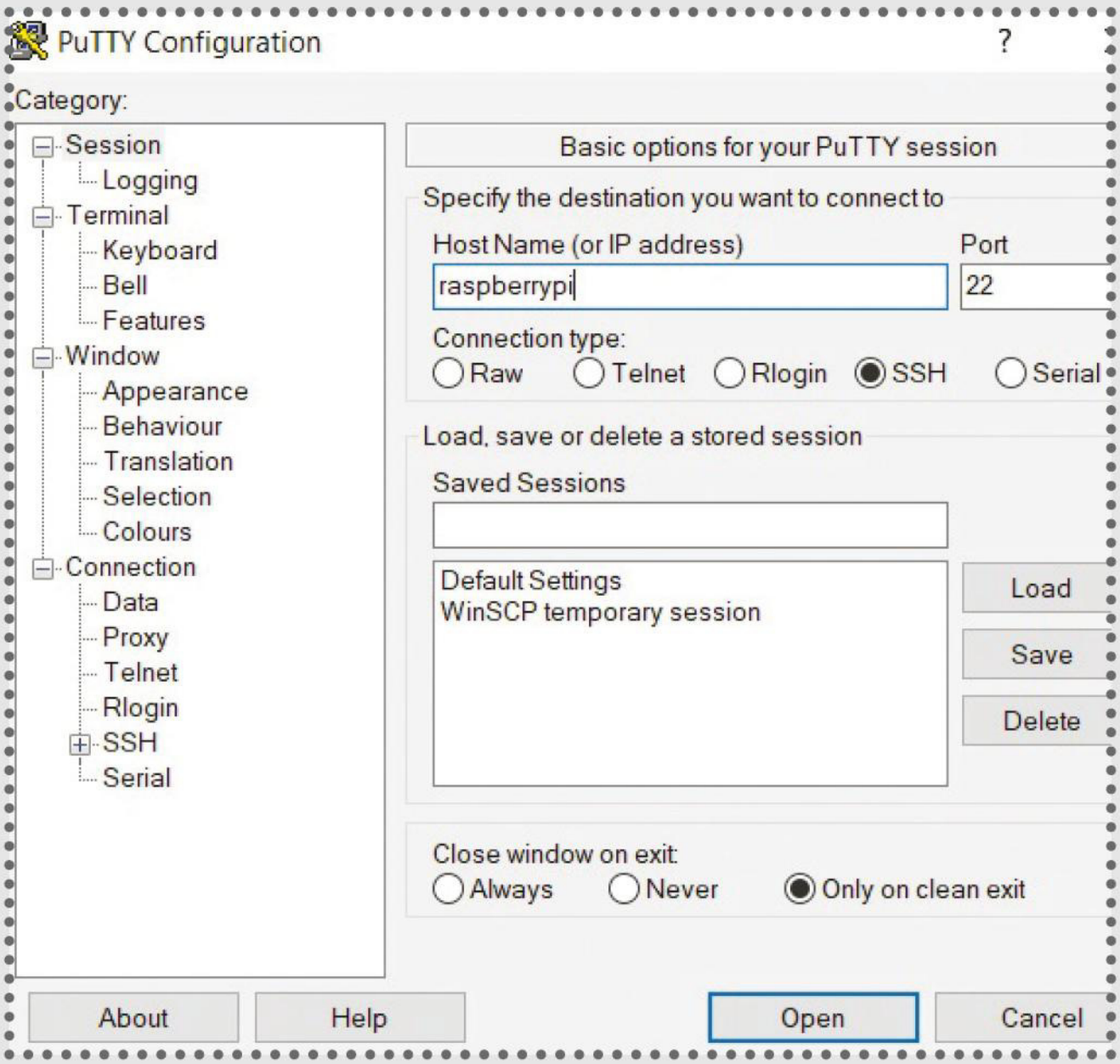
03 Set up the Wi-Fi

The Pi Zero is the perfect size for embedding into projects and hacks. When using it within clothing it is unlikely that you will have a display or screen available. So a simple solution to access your Pi is via SSH (Secure Shell) This requires Wi-Fi access. An easy way to set this is to hook up the Pi Zero to a monitor, boot it up, select the Wi-Fi icon in the top right-hand corner and enter the 'pre shared key' for your wireless router. Save the settings and each time your Pi

Zero starts up, it will attempt to connect to your network.

04 SSH into the Pi Zero

Now the Wi-Fi is operational, download and install an SSH client such as PuTTY onto your laptop. Run PuTTY, enter the IP address of your Pi Zero (usually you can use the default name **raspberrypi**). Enter your username and password when prompted and you will be presented with the Terminal window. This can be used to write code, run code and set up the project. The other option is to create the code on a Raspberry Pi 2 or 3 using a monitor. Once you have set it up, tested it and got it working, transfer the card into the Pi Zero W and embed the hardware into the Christmas jumper.



05 Install Tweepy

You have now registered your Twitter app, acquired the consumer keys and tokens, and have access to your Pi. The next step is to download Tweepy, the Python Twitter API.

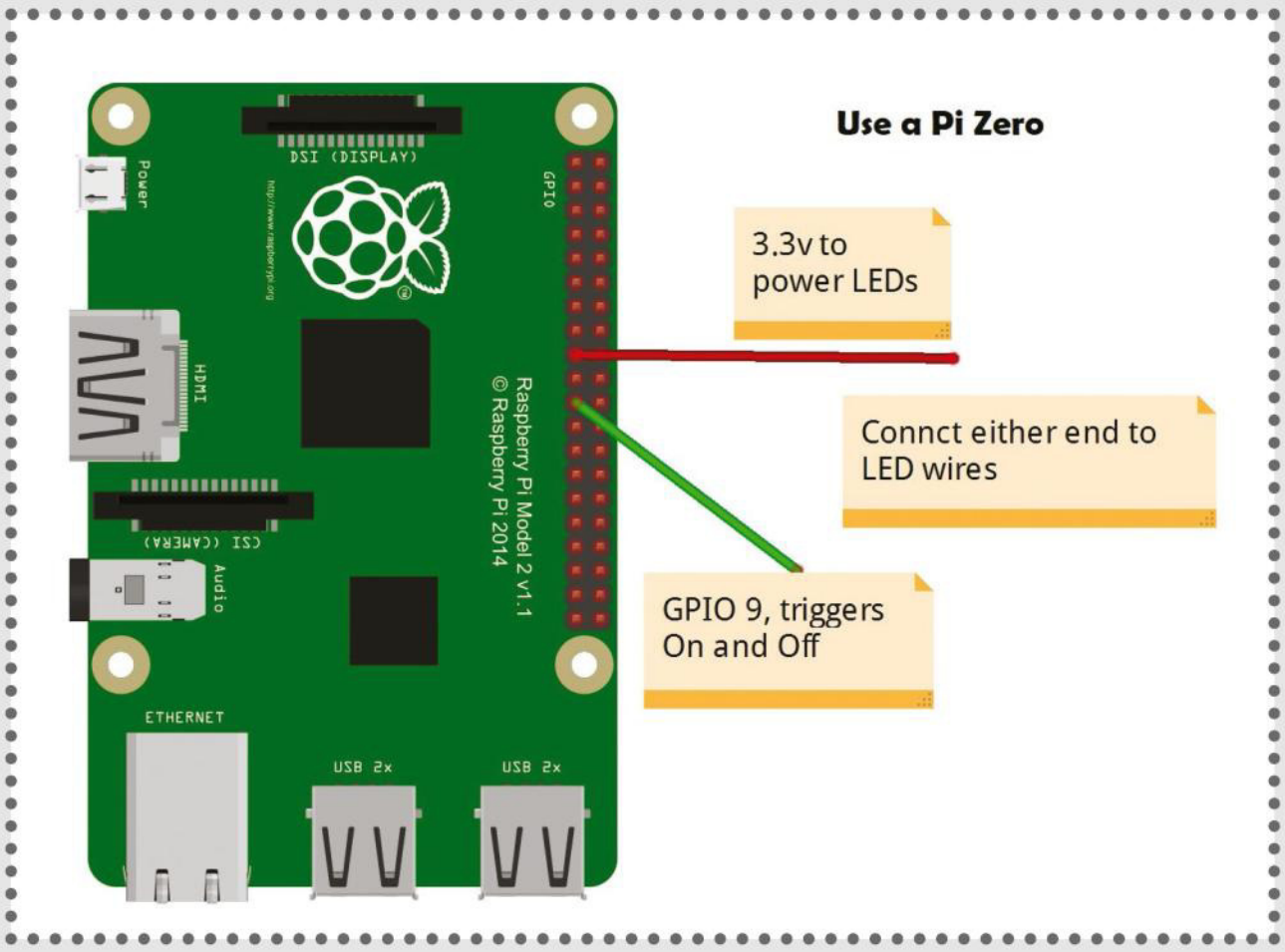
In the Terminal, type:

```
sudo pip3 install tweepy
```

Reboot your Pi Zero W by entering sudo reboot. You will then need to SSH in again. You can now use Python code to interact with your Twitter feed.

06 Prepare the LEDs

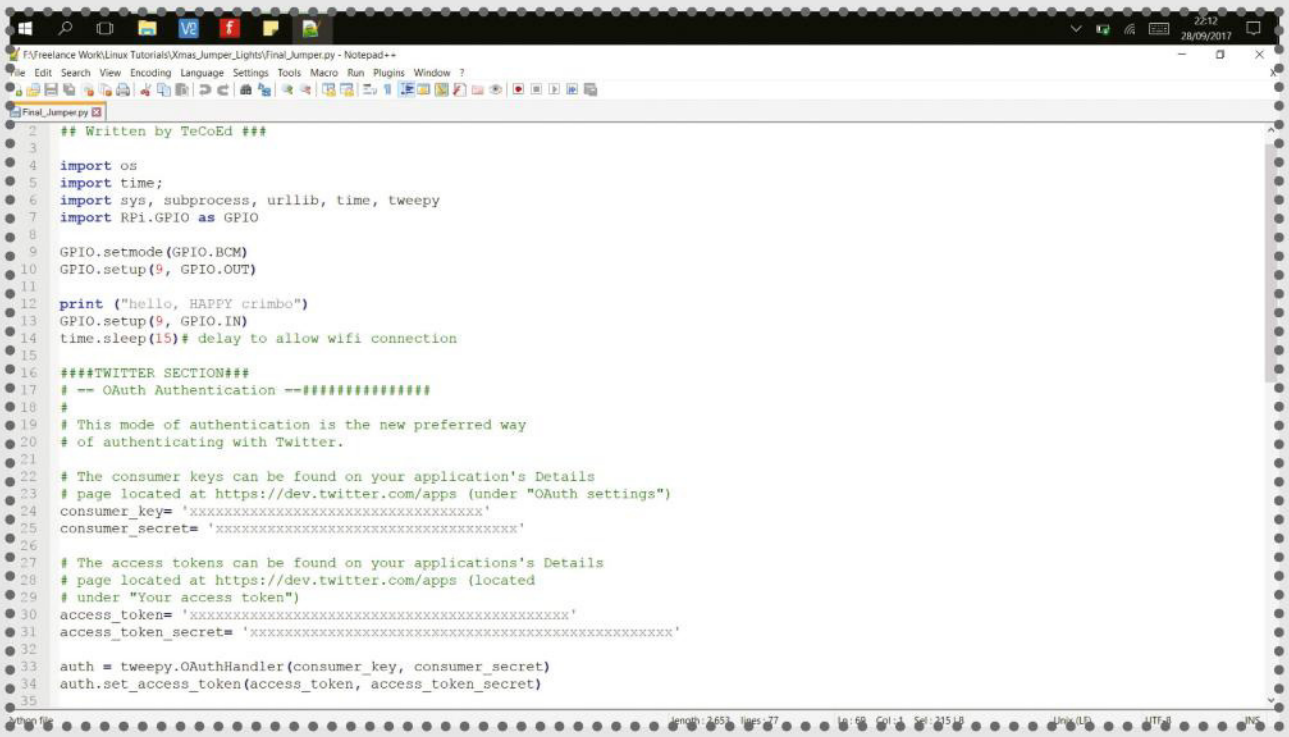
There are two ways to power the LEDs. Take one of the wires and cut it in half between the LED and the battery pack. Solder to each end a male-to-female jerky wire. The Pi Zero will provide either 3.3V or 5V. Attach one of the wires



to the LEDs via, for example, GPIO pin 2, 4 or 9 and then the other wire to another pin, GPIO 11. Or use a non-power pin and a ground pin, the power for the LEDs being provided by the batteries. It is not advisable to use both the battery power and the Pi as this will quickly burn out the LEDs.

07 Save images and take a sensor reading

Now you have Twitter and the LEDs ready, open a new Python file. Import the modules to enable the Python program to interact with Twitter and control the LEDs. The final line of code enables a 15-second delay at the start of the program. This is to ensure that the Pi Zero W has sufficient time to establish a Wi-Fi connection before the main program attempts to communicate with Twitter. Otherwise you will receive an authorisation error.




```
import os
import time;
import sys, subprocess, urllib, time, tweepy
import RPi.GPIO as GPIO
```

```
  GPIO.setmode(GPIO.BCM)
  GPIO.setup(9, GPIO.OUT)

  print (“hello, HAPPY crimbo”)
  GPIO.setup(9, GPIO.IN)
  time.sleep(15)# delay to allow wifi
connection
```

08 Authorising a Twitter connection

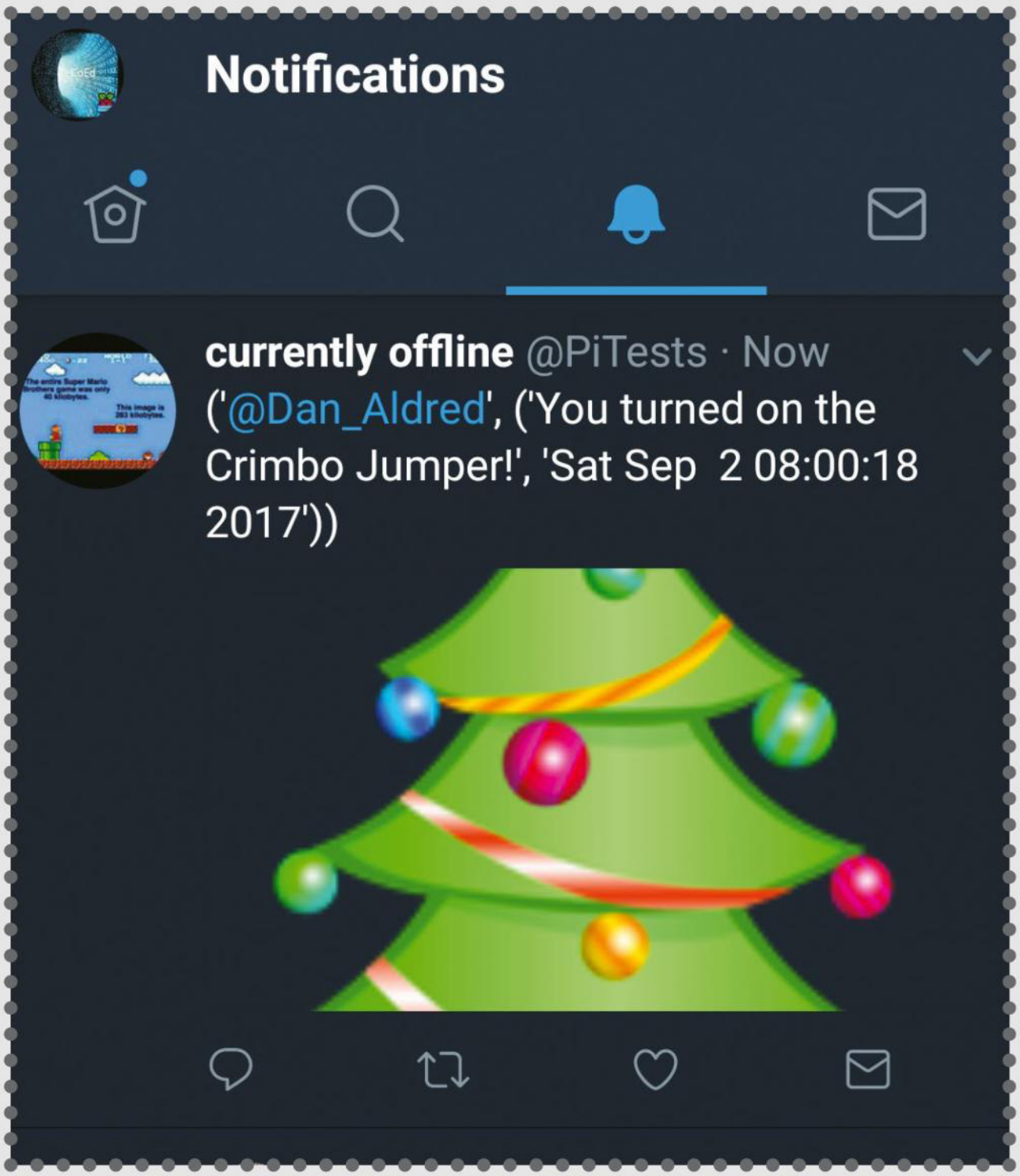
To stream your tweets, you need to authorise a connection using your Twitter consumer keys and the access token that you set up in **Step 2**. Add the lines of code below, which will establish the connection and enable you to stream and send tweets.

```
  ####TWITTER SECTION###
  # == OAuth Authentication ==################
  consumer_key= ‘xxxxxxxxxxxxxxxxxxxxxxxxxxxx’
  consumer_secret=
‘xxxxxxxxxxxxxxxxxxxxxxxxxx’

  access_token=
‘xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx’
  access_token_secret=
‘xxxxxxxxxxxxxxxxxxxxx’

  auth = tweepy.OAuthHandler(consumer_key,
consumer_secret)
  auth.set_access_token(access_token, access_
token_secret)

  api = tweepy.API(auth)
```

09 Check tweets for a keyword – Part 1

Add the main part of the program to check for the 'trigger' word. This ensures that a user meant to turn on the jumper's LEDs, rather than by accident. We define a class to hold the code, then collect each tweet and store it in a variable called tweet_to_check.

```
class Crimbo_Lights(tweepy.StreamListener):
    def on_status(self, tweet):
        tweet_to_check = tweet.text ##gets tweet
```



10 Check tweets for a keyword – Part 2

We print out each tweet, line one. Line two looks for your exact keyword in the Tweet and returns the numerical position of the word. This will always be zero as the phrase starts from position zero. If the value is not zero, then the exact phrase has not been sent in the message.

```
  print (tweet_to_check)
  does_the_tweet_contain_key_word = tweet_to_
check.find("@XXXXX ON")
  print (does_the_tweet_contain_key_word)
```

11 Add a festive picture and message to the tweet

Next, the program needs to check if the trigger word value is zero. An if statement compares the two values, line one. Set the GPIO board configuration and the GPIO pin number for the output, lines three and four. Source and save a suitable festive image that you wish to send in your reply to the user. Make a note of the file path of the image, then create new variable to store this path. In this example, the image is stored in /home/pi/ and the variable is called pic.

```
  if does_the_tweet_contain_key_word >= 0:
   GPIO.setmode(GPIO.BCM)
   GPIO.setup(9, GPIO.OUT)
   pic = '/home/pi/lights.jpg'
```

12 Get username

Retrieve the user's Twitter ID, line one. Doing this enables you to send the user a message notification confirming that they have triggered the Christmas jumper LEDs. However, if you send the same tweet more than

Headless Pi Wi-Fi setup

You can set up the Wi-Fi connection via the SD card before you boot up your Pi. This involves a few more steps than the GUI interface but will give you some understand of the different elements of information required to connect to the Wi-Fi. Check out this Forum guide form the Raspberry Pi website: **http://bit.ly/HeadlessWiFi.**

once, Twitter won't permit the message to be sent. To work around this, record the time the user triggered the jumper LED lights and add the time to the message. This makes each tweet unique and they appear in real time.

```
  user = str(tweet.user.screen_name)
  time_sent = time.asctime( time.
localtime(time.time()) )
```

13 Combine the parts of the message

Create a variable called message and add your festive response. Then combine this with the user ID from the previous step to create your final message.

```
  message = "You turned on the Crimbo
Jumper!", time_sent
  final_tweet = "@%s" %(user), message
```

Final_Jumper.py

```
### CODE TO GET TWITTER TO LISTEN FOR PHRASE ###
class Crimbo_Lights(tweepy.StreamListener):
    def on_status(self, tweet):

        tweet_to_check = tweet.text ##gets the tweet
        print (tweet_to_check)

        ###Checks for tweets to your Twitter ID @xxxx
        does_the_tweet_contain_key_word = tweet_to_check.find("@XXXXX ON")
        ###change to use code to find key word###
        print (does_the_tweet_contain_key_word)

        if does_the_tweet_contain_key_word >= 0:
                GPIO.setmode(GPIO.BCM)
                GPIO.setup(9, GPIO.OUT)
                pic = '/home/pi/lights.jpg'
                user = str(tweet.user.screen_name)
                time_sent = time.asctime( time.localtime(time.time()) )
                message = "You turned on the Crimbo Jumper!", time_sent
                #print error_tweet
                final_tweet = "@%s" %(user), message
                #print type(error)

                ###Turn the lights on
                GPIO.output(9, GPIO.LOW)
                time.sleep(8)
                ###Turn them off
```

14 Turn the lights on and send your message

Now to turn your Christmas Jumper lights on. Set the GPIO output to LOW so that the circuit is connected and the battery powers the lights. Next, add a delay for how long you want the lights to stay on – in this example program, 8 seconds. Then turn the lights off. Finally, combine your festive image from Step 11 and the reply message from Step 13 using the code api-update_with_media, to tweet your message.

```
###Turn the lights on
GPIO.output(9, GPIO.LOW)
time.sleep(8)
###Turn them off
GPIO.setup(9, GPIO.IN)
api.update_with_media(pic, final_tweet)
```

15 Streaming and checking for tweets

The final piece of the program code responds to tweets not containing the trigger word. Add an else statement (take care with the indentation levels), then print an optional short notification that the lights were not triggered. Set the GPIO to IN, line three, ensuring that the lights stay off. Finally, set the tweets streaming from your timeline using a while True loop.

```
else:
print (“no lights”)
GPIO.setup(9, GPIO.IN)

stream = tweepy.Stream(auth, Crimbo_Lights())
```

Tether your Pi to your mobile phone

The Pi Zero can be tethered to a mobile phone to ensure that the Christmas jumper is mobile-interactive when you’re out and about. To maintain the connection, it is advisable to disable the Wi-Fi management. To set this up, in a Terminal type: sudo nano /etc/network/interfaces. Then add the line: wireless-power off. Save the file and reboot the Pi.


```
while True:
    stream.userstream()
```

16 Automatically start the program

Since the program is running on a Pi Zero W and embedded into the jumper, it needs to start automatically when the power is turned on. To enable the program to run at startup, enter: sudo nano /etc/rc.local in Terminal. At the bottom of the file, add the file path of the program; e.g., python /home/pi/LED_Hat.py & – don't forget the & symbol at the end. Reboot the Pi Zero and the program will start automatically. Pop your Christmas jumper on and get your followers, friends and other festive people to tweet you!

Raspberry Pi tablet

The elegant, streamlined design yields probably the smallest and lightest homemade Pi tablet in existence

With this project, Stefan Vorkoetter takes the term ‘Pi hack’ and applies it quite literally to hacking down a Raspberry Pi 3 Model B to the bare essentials required for the job, combining it with a 7-inch display to create a lovely handmade Pi tablet sat in a maple frame. His streamlining efforts reduce a recommended thickness of 40mm for the combined Pi and display down to 17.6mm and include so many intelligent touches: an aftermarket USB audio adaptor for use with earbuds or speakers; a real-time clock; and a modified Adafruit PowerBoost 1000 charger with homemade heatsinks, shutdown controller and precision battery monitoring.

You added a number of components that were missing from the Pi, as well as stripping away a lot of things – is there anything on your wishlist for the next Pi?

Number one would probably be more RAM. Don’t get me wrong, the Pi works much better with 1GB than my Windows-based travel notebook did with 2GB (I’ve since put Linux on it), but 2GB would be a big improvement.

The other thing that would really have helped this project is some sort of ability to suspend to RAM or disk. Right now, the tablet has to be booted to use it, and that takes about 28 seconds, which is a long time if you just want to check the weather. There are also some glitches that need fixing, like the inability to use multiple Bluetooth devices when the built-in Wi-Fi is active.

You seem to have tackled the power issues comfortably whereas similar projects have

Stefan Vorkoetter is an R&D Fellow with Maplesoft, which is a Canadian mathematics and simulation software development company. One of his background projects is a single-seat plane.

struggled. Why is that?

I think there are several factors. One is probably just that I've been doing this longer. I've been into electronics as a hobby for a very long time, [learning from his father and Mark Tilden, renowned robot physicist, during his years at the University of Washington] whereas a lot of the projects I read about seem to be by relative newcomers to the field. I've had a lot more time to learn from mistakes. I think the other big help is that I'm not afraid of the analogue realm, having designed and built a lot of purely analogue stuff. [...] I'll often do things with a few op-amps and resistors that others might throw an Arduino at.

Was the homemade battery monitor and shutdown controller part of the initial plan or did you realise it was something you needed for the PowerBoost 1000 charger later on?

I knew I'd need battery monitoring, and somehow allow for an orderly shutdown, although I wasn't completely sure how when I first started. After I received the PowerBoost 1000C, I did some experimentation to help develop the ideas for the shutdown aspect of it.

For the battery monitoring, I had actually settled on, and built, a slightly more complex circuit that used a bigger chip (quad op-amp) and more discrete components, but only gave four data points (75%, 50%, 25%, 12.5%). But then one day, probably while driving home, I realised I could do a better job with half the parts. The 'Ah ha!' moment was when I realised that I didn't need instantaneous A/D conversion for something that changes as slowly as battery voltage. The final solution uses very slow

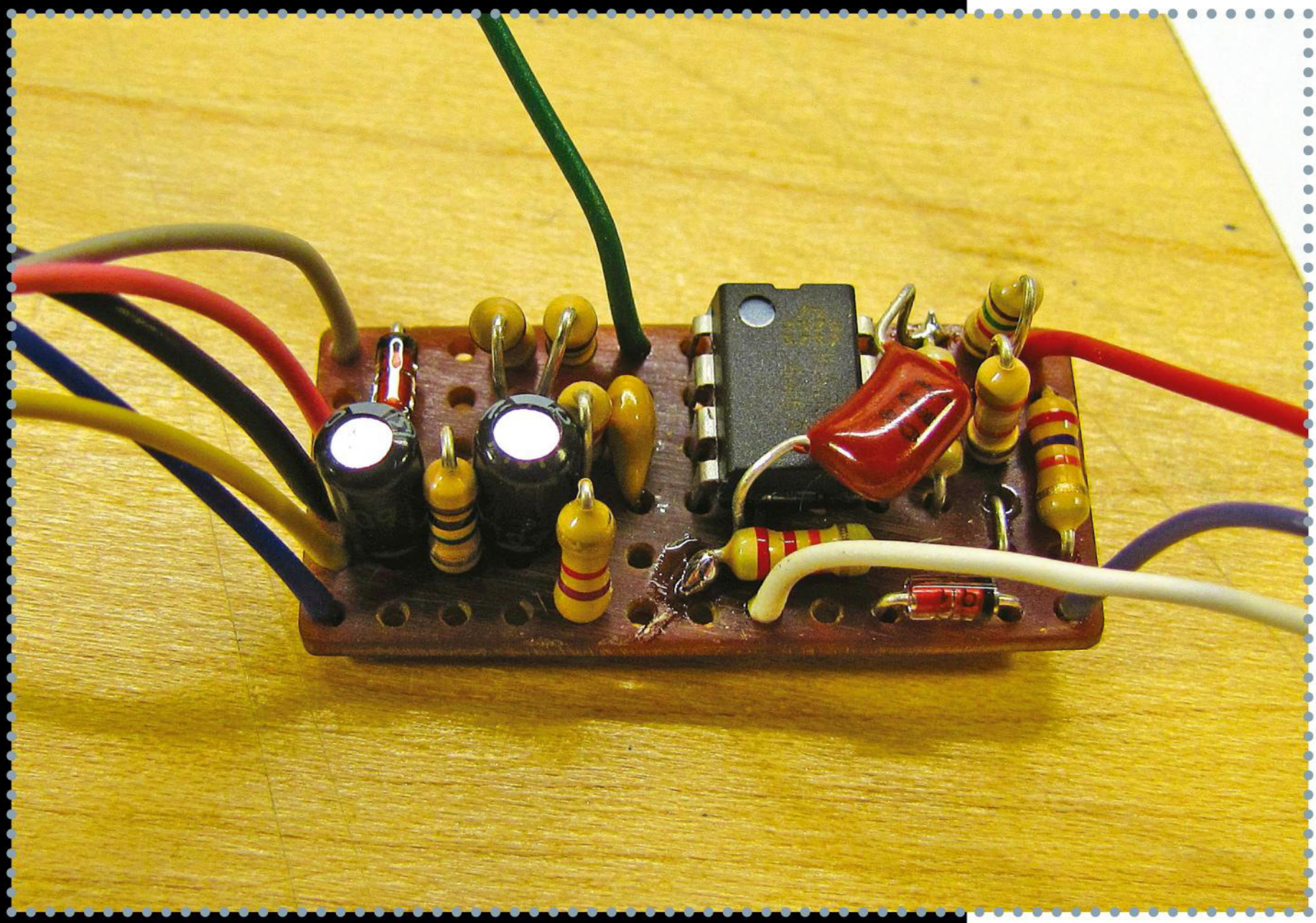
THE PROJECT ESSENTIALS

- Raspberry Pi Model 3B
- Raspberry Pi 7-inch 800x480 multi-touch capacitive touchscreen
- Lexar 32GB microSD card
- Adafruit PowerBoost 1000C charger and 3.7-to-5V converter
- DS3231 temperature-compensated real-time clock
- USB audio adaptor
- PAM8302-based mono class D amplifier
- 25mm 8 speaker
- Battery monitor and shutdown control circuit
- 6200mAh Lithium-polymer battery, giving 4 to 12 hours battery life (6 hours watching YouTube)
- SPDT slide switch and 3× small push-button switches
- USB and micro-USB ports
- Raspbian Jessie

PWM to give me sub-millivolt precision but requires 16 seconds to get an initial reading, after which I get continuous updates that are 8 seconds out of date.

Can you tell us about some of the code for the project?

There are basically two pieces that I wrote. A daemon, started at boot time, runs in the background, monitoring the battery voltage, buttons and power switch. It controls the display brightness, brings the on-screen keyboard (Tom Sato's xvkbd) to the front, maximises apps, and communicates status with the dashboard program. The dashboard displays the current battery level and core temperature, and also



Below There are lots of clever touches in the Pi tablet's design; one was the decision to add homemade heatsinks to the charging and power conversion chips on the PowerBoost 1000 charger

lets the user interact with the daemon to turn devices on and off (to save power) and control automatic screen dimming.

The daemon is written in C, since that is what one writes daemons in, and I've been doing UNIX C programming since the mid-1980s. I'd never actually written a daemon before, but it's pretty easy these days.

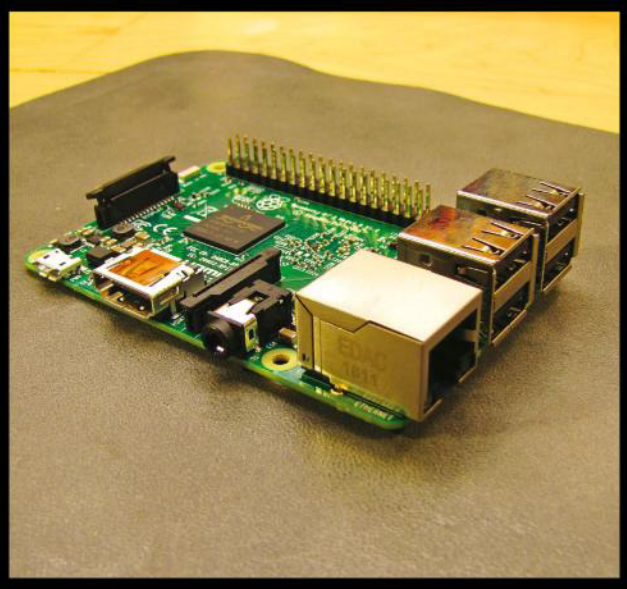
The dashboard is written in Free Pascal using Lazarus, which is very similar to Borland's Delphi that I've used for various other (non-work) projects that required GUIs in the last few decades (e.g. MotoCalc, Watch-O-Scope).

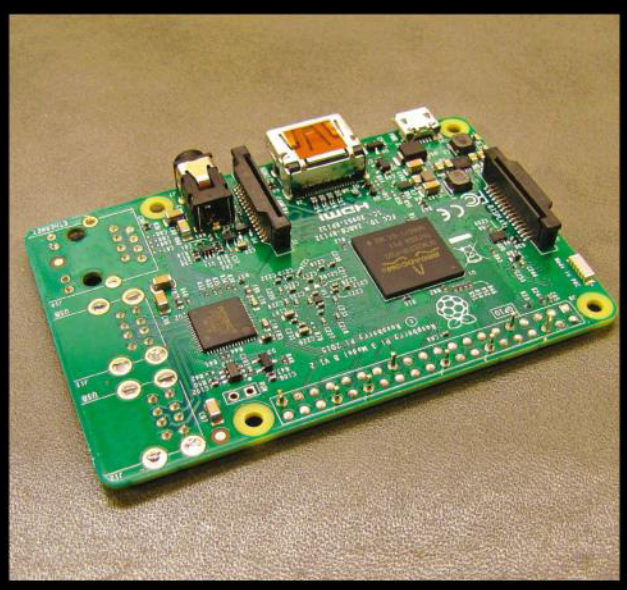
Like it?

Stefan has two projects on the go: a Pi-based music synthesiser (built into the top of his Hammond organ); he's also working on improving the aerodynamics of his car for better fuel efficiency and says he'll "probably end up putting a Pi in there as well…"

Below Vorkoetter overcame two power issues by designing a small circuit containing both a battery voltage monitor (see the annotation, above) and a shutdown controller. You can read his detailed explanation of the whole project at **http://bit.ly/PiTablet**

Would you do things a little differently next time?
One issue is if I were to forget to turn it off. The daemon will eventually do a shutdown when the PowerBoost's low battery output is triggered, but even a shutdown Pi will continue to consume about 200mA. The safety circuit in the battery will cut the power completely at 2.9V, but that's too low to be good for the longevity of the battery. I've designed another tiny analogue circuit that will tell the PowerBoost to cut the power before it reaches that point, and will probably tuck it into the empty space between the two USB ports. If I were to redo the project, I'd incorporate that into the existing power monitor board [...]. I'd probably also consider 3D printing an enclosure for next time.



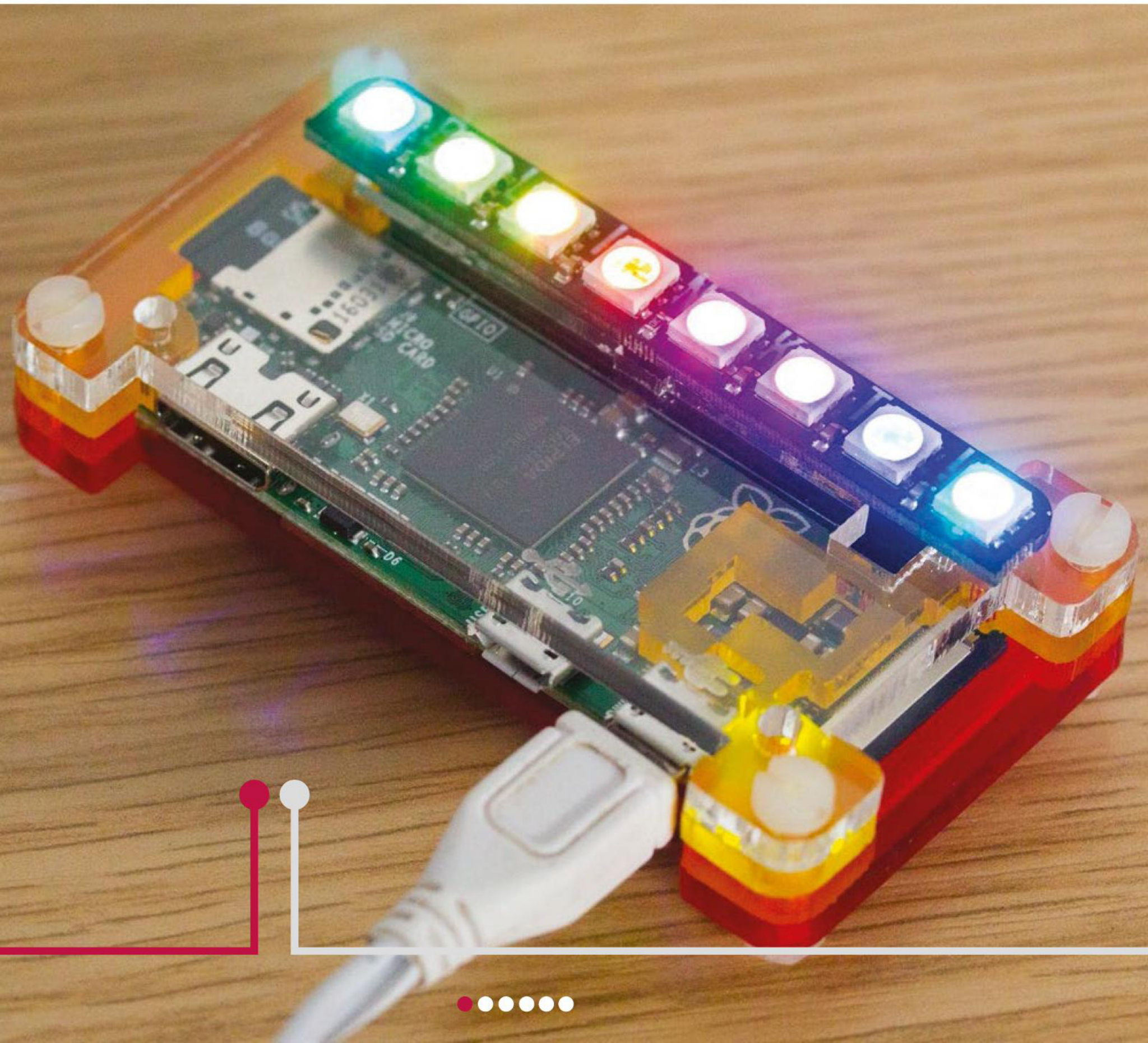


Further reading

Stefan's recent projects include customised mechanical wristwatches, a remake of an analogue music circuit from the 1970s for an alarm clock and a complete rebuild of the vacuum tube (valve) amplifier in his 1962 Hammond organ. Read more at **www.stefanv.com.**

Get hands on with the Pimoroni Blinkt!

Learn the basics and create sparkling lights with your Blinkt!



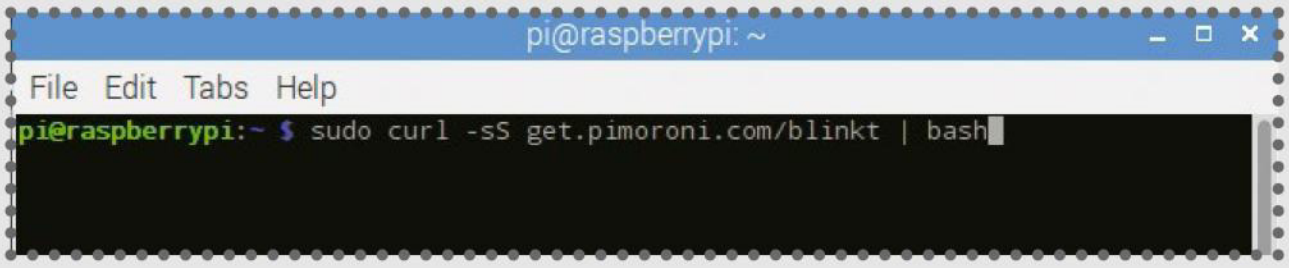
Pimoroni has created an awesome 'super-bright RGB LED indicator' that is ideal for adding visual notifications to your Raspberry Pi without breaking the bank! The Blinkt! offers eight APA102 pixels in the smallest (and cheapest) form factor to plug straight into your Raspberry Pi 3/2/B+/A+/Zero. This tutorial walks you through how to install the Blinkt! and the required Python libraries and modules. Next, create and try out some simple code to control the lights and change the colours and the brightness of the LEDs. Finally, combine these skills together to code an LED colour generator that selects a random LED and a random RGB colour value for the range of eight lights. This creates a psychedelic disco-light setup.

01 Install the Blinkt!

The guys at Pimoroni have made it very easy to install the software for your Blinkt! (which also includes a wide selection of cool examples to show off its features). To get started, ensure that the power is off, attach the Blinkt! to the Raspberry Pi's GPIO pins and push down firmly. One side of it is straight and one is curved. The curved edges align with the curved corners of your Raspberry Pi. This ensures that you attach it the correct way round. Now attach your power supply and boot up your Pi. Open the Terminal and type:

```
curl -sS get.pimoroni.com/blinkt | bash
```

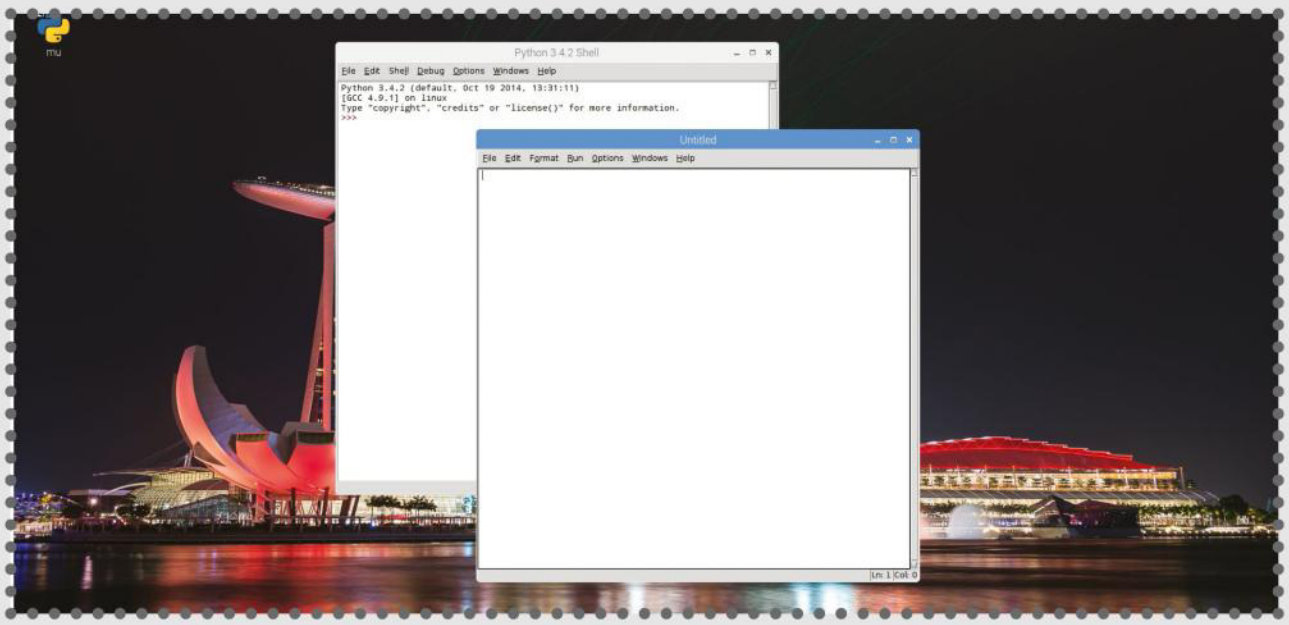
This will install all the required code.



02 Turn on an LED

To turn on an LED, open your Python editor and import the required Python module, line 1. Use the code set_pixel(), line 2, to set which pixels are turned on. The first number in the brackets corresponds to each LED. As with the common computing method of numbering, the first LED is referred to by the number zero, the second LED is number one and so on up to the eighth LED, number seven. Copy the code below and run the program to turn the first LED on.

```
from blinkt import set_pixel, show
set_pixel(0,255,0,0) show()
```



03 Change the colour of an LED

The next three values in the brackets (x, 255, 210, 150), are used to control the colour of the LED with the standard RGB colour palette. You can control and set the amount of Red, Green and Blue between the values of 0 and 255; the higher the number, the more of that colour is displayed on the LED. Combining these values gives you over 16 million possible combinations and colours. Change the values in your code, save and then run.

```
set_pixel(0,255,255,0) show()
```

04 Adjust the brightness of the LEDs

You may find that the LEDs are too bright to look at. It is not advisable to look at them for long periods as they may damage your eyes. Instead, turn down the brightness so that they can be viewed safely. First, import the brightness module, line 1. You can import multiple modules by including the 'module name' on the same line– for example, from blinkt import set_pixel, show, set_brightness. Now set the brightness to a suitable level, line 2. The values range between zero and one, where one is full brightness and zero renders all the LEDs off.

```
from blinkt import set_brightness
set_brightness(0.5)
```

05 Turn on multiple lights

Turn multiple LEDs to on using the set_pixel code and then set the RGB colour values. Remember that for the LED number, 'zero' is the first LED, as you count from zero upwards. The last LED is number seven. Try turning on two or more LEDs; for example, the last LED set to red and the fourth LED set to blue. Save and run the program.

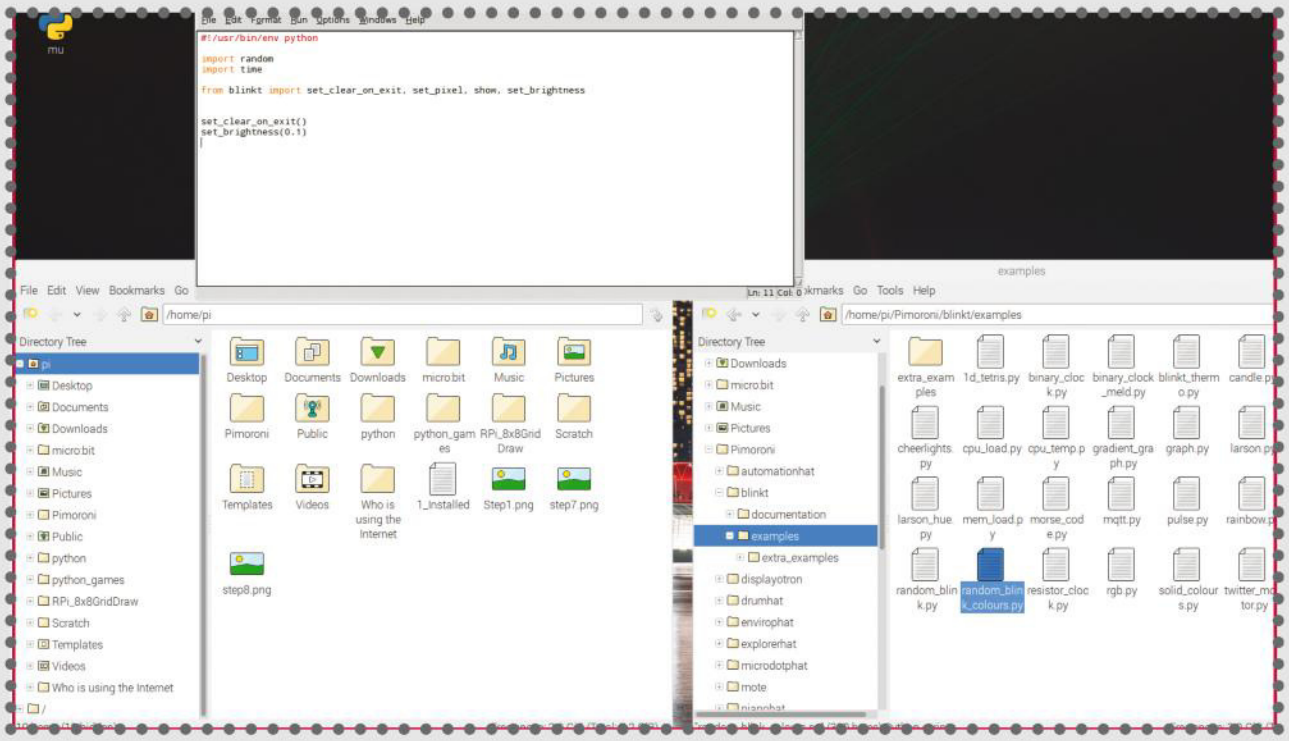
```
set_pixel(7,255,0,0) show()
set_pixel(3,0,0,255) show()
```

06 Create a random blinking light

Combine the previous code steps and create a random set of sparkling disco lights! Begin by selecting a new

Who is using the internet?

Do you live in a household or work in an office where several people use the internet? Are there numerous devices always connected to the network that suck the bandwidth up and slow down the speed for all other users? Combine the Blinkt with Python and nmap to identify users, track them on the network and create a colourful LED visual for each user as they join and leave the network. Watch the video here: **youtu.be/r_JDw65FTMA**



blank Python file and importing the random module, line 1. This is used to select random values for the colours and the LED position number. Next, import the time module, line 2. This adds control over the frequency of the change (a bigger time value will result in slower blinks). Finally, import the codes to clear the LEDs when the program exits, set a particular pixel to on and reduce the brightness, line 3. Line 4 enables the LEDs to clear on exit and line 5 sets your required brightness value between zero and one.

```
  import random
  import time
  from blinkt import set_clear_on_exit, set_
pixel, show, set_brightness
  set_clear_on_exit()
  set_brightness(0.1)
```

07 Create a while loop and set random values

Now create a while loop to ensure that the program continually repeats and loops over the code, line 1. On the next line, (which is indented), use a for loop to iterate

the code over each of the LEDs. This has the effect of applying the next instruction to each of the LEDs, working from the first LED to the last, line 2. Turn the LED on using the set_pixel code, line 3; note that this is indented too.

```
  while True:
    for i in range(8):
      set_pixel(i,
```

08 Select a random colour

When coding the LED, state the RGB colour values of the LED as covered in step 3; use the code random.randint(0,255). This randomly selects a colour value between 0 and 255. Add two more incidences of the code to select values for the Green and Blue colours. On line 2, set the LEDs to display the colour using the code show(); note that this line doesn't have an indent. Finally, add a short time delay so that you can observe the lights before they change state, line 3.

```
    set_pixel(i, random.randint(0,255), random.
randint(0,255), random.randint(0,255))
  show()
  time.sleep(0.05)
```

09 Run the program

Save your program file and run it by pressing F5; call the file a suitable name. Press Enter and the program will run, displaying a variation of random colours on each of the LEDs. Experiment with the colour values to create variations that you like and also to speed up or slow down the delay between changes.

Red, Green and Blue colours

The RGB colour model is an additive colour model in which red, green and blue light is combined in various amounts to create a wide range of colours, over 16 million different variations. The lowest value for a colour is zero, which usually denotes black, as black is not a colour but an absence of colour! The top value is 255; for example, red 255 means the maximum amount of red is being added to the shade.

Hack Minecraft on a friend's Raspberry Pi over a network

From one Raspberry Pi to another, have some fun with a friend's Minecraft world over a local network

We will create a Python script that connects directly to a Minecraft game running on another Raspberry Pi, which will enable us to have some fun with their game world. We'll have the ability to manipulate the character, the environment, and place blocks, as we've done in the previous tutorials, but this time we're working on someone else's game. We'll be able to have pranks galore, but it should be mentioned that this should not be done without prior permission of the third party.

This tutorial is written under the assumption that you're running Minecraft Pi Edition on a Raspberry Pi. No additional software is required. If you'd like to run this tutorial on your own flavour of desktop Linux, we've also put together a number of tools to ensure this hack works for you, Pi or not, with a retail version of Minecraft. To get your retail Minecraft interacting with Python, you'll need to install McPiFoMo by copying the contents of the .minecraft directory into ~/home/.minecraft. McPiFoMo includes MCPiPy from MCPiPy.com and Raspberry Jam, developed by Alexander Pruss.

```
1   import mcpi.minecraft as minecraft
2   mc = minecraft.Minecraft.create()
3   friendsIP = "192.168.15.15"
4   hackedPi = minecraft.Minecraft.create(friendsIP)
5
6   hackedPi.postToChat("Hello world!")
7
8   hackedPos = hackedPi.player.getTilePos()
9   x = hackedPos.x
10  y = hackedPos.y
11  z = hackedPos.z
12
13  hackedPi.setBlock(x, y, x, 35, DIAMOND_ORE)
14  #hackedPi.postToChat(hackedPos)
15  #hackedPi.player.setTilePos(10,10,10)
16
```

Python scripts should be saved in ~/home/.minecraft/mcpipy/, regardless of whether you're running Minecraft Pi Edition or Linux Minecraft. Be sure to run Minecraft with the 'Forge 1.8' profile included in McPiFoMo.

01 Getting your friend's IP

Before we do anything, we'll need to know the IP address of our friend's Raspberry Pi. Make sure your Pi is connected to the same network as theirs, and run Angry IP Scanner. This will list all the computers connected to the same network as you, within your IP range by default. Look for a hostname that suggests a Raspberry Pi. If your friend is running Raspbian, the default hostname will be 'raspberrypi'. Be sure to identify your own IP for exclusion, by opening a Terminal and running ifconfig /all.

02 Initiate a Python script

Create a new Python script in IDLE or your favourite text-based editor:

```
import mcpi.minecraft as minecraft
mc = minecraft.Minecraft.create()
friendsIP = "192.168.1.2"
hackedPi = minecraft.Minecraft.create(friendsIP)
```

This time, we're also start a connect to our friend's IP.

03 Hello World!

As with any programming tutorial, we start off with a quick 'Hello World!':

```
hackedPi.postToChat("Hello world!")
```

Save this script in your ~/home/.minecraft/mcpipy/ directory with a name like hax.py and then run the script directly from Minecraft Pi by typing '/python hax' in the chat window and pressing Enter. You'll notice your friend's game has now displayed 'Hello World!'. May the fun commence!

04 Placing blocks around our friend

Now that we've connected and communicated with our friend's game, it's time to start building something around them. We'll need to gather their player position and place blocks relative to them:

```
hackedPos = hackedPi.player.getTilePos()
hackedPi.setBlock(hackedPos.x,hackedPos.y,hackedPos
.z,block.DIAMOND_ORE)
```

Now that we have our friend's player position, we can build around them by altering the x,y,z coordinates and block type accordingly.

05 Building in their world

To take creations from previous Minecraft Pi tutorials and convert them to work across the network, we'd replace the mc variable which points to our game world, with hackedPi referring to our friend's.

```
mc.setBlock(blockX, blockY, blockZ, woolBlockBlack,
    woolBlockBlackType)
```

And this becomes:

```
hackedPi.setBlock(blockX, blockY, blockZ, woolBlockBlack,
    woolBlockBlackType)
```

06 Hack pixel-art Creepers into friend's game (Part 1 of 3)

Let's start a new script to convert our Creeper head from last issue. Initialise the connects (as above, in Step 2) and create some new variables:

```
woolBlockGreen = 35
woolBlockGreenType = 5
woolBlockBlack = 35
woolBlockBlackType = 15
```

07 Hack pixel-art Creepers into friend's game (Part 2 of 3)

Create our pixel art with alternating block types.

pixelArt = [[1, 1, 1, 1, 1, 1, 1, 1],[1, 0, 0, 1, 1, 0, 0, 1],[1, 0, 0, 1, 1, 0, 0, 1],[1, 1, 1, 0, 0, 1, 1, 1],[1, 1, 0, 0, 0, 0, 1, 1],[1, 1, 0, 0, 0, 0, 1, 1],[1, 1, 0, 1, 1, 0, 1, 1],[1, 1, 1, 1, 1, 1, 1, 1]]

In the next step, we'll link these 1s and 0s to our previously initialised variables, assigning 0 to woolBlockBlack and 1 to woolBlockGreen. When we run this code, we'll spawn a large pixel-art Creeper head in front of our friend's player.

08 Hack pixel-art Creepers into friend's game (Part 3 of 3)

```
  pos = hackedPi.player.getTilePos()
  for row in range(len(pixelArt)):
      for pixel in range(len(pixelArt[row])):
        if pixelArt[row][pixel] == 0:
            hackedPi.setBlock(pos.x, (pos.
y+7) - row, pos.z + pixel, woolBlockBlack,
woolBlockBlackType)
        elif pixelArt[row][pixel] == 1:
            hackedPi.setBlock(pos.x, (pos.
y+7) - row, pos.z + pixel, woolBlockGreen,
woolBlockGreenType)
```

09 Spawn it, blow it

Save your new script in ~/home/.minecraft/mcpipy/ and run it directly in Minecraft Pi with '/python scriptname'. Now sit back and watch your friend jump when a giant Creeper head appears in front of them.

If you want to take things to the next level, you could duplicate the for loop to spawn rows of TNT behind the Creeper head. An explosive Creeper head would be quite something. The easiest way to spawn primed TNT

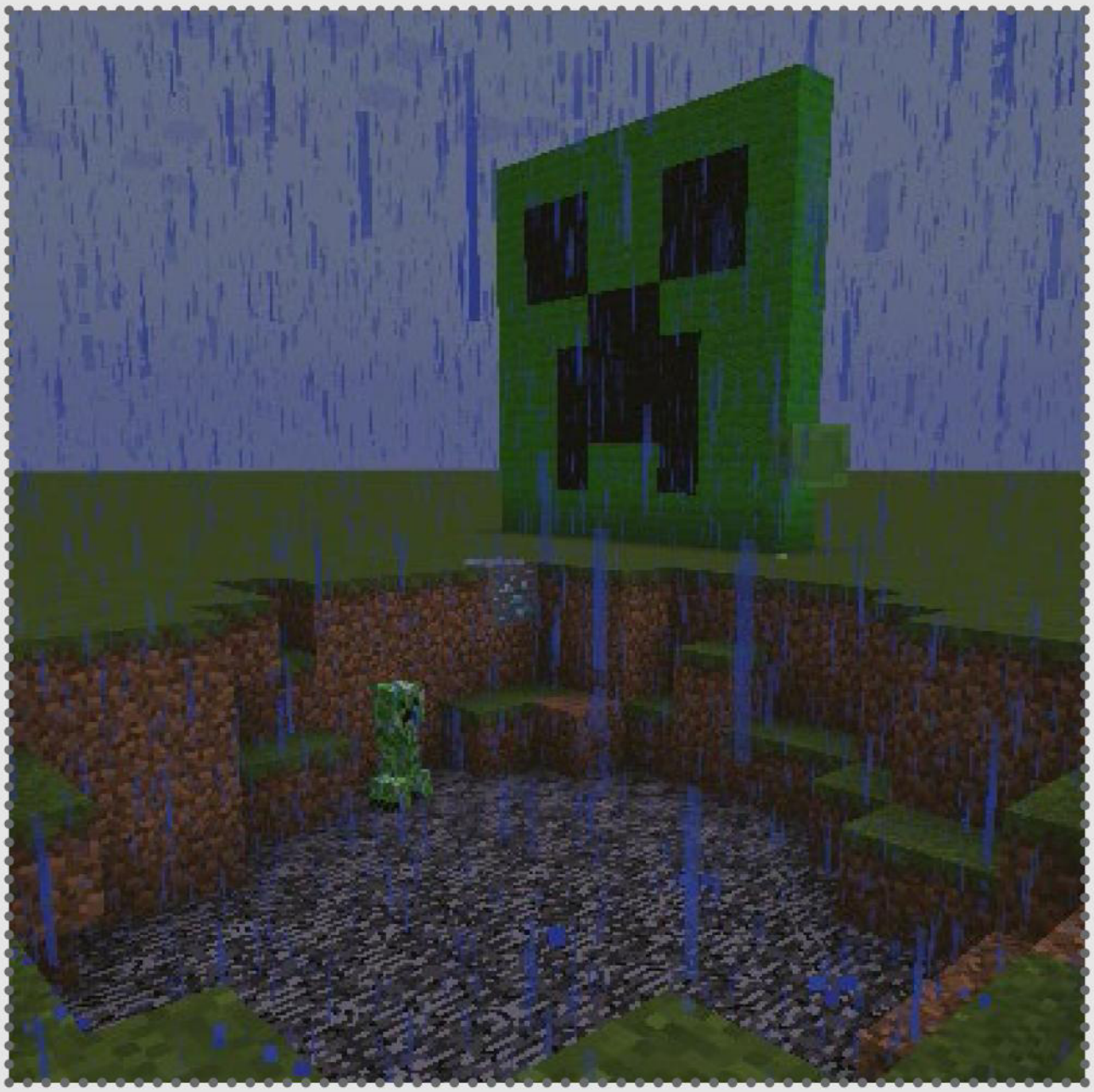
is to place it next to an enabled redstone torch (blockID 76).

10 Teleport your friend around their world

We've controlled the game world by placing blocks around our friend's player, and we've communicated with them directly by displaying text on their screen. Another fun way of messing with our friend is by teleporting their player around their world.

```
hackedPi.player.setTilePos(x,y,z)
```

Initialise some the x,y,z variables and set them to whatever coordinates you want your friend's player to be teleported to.

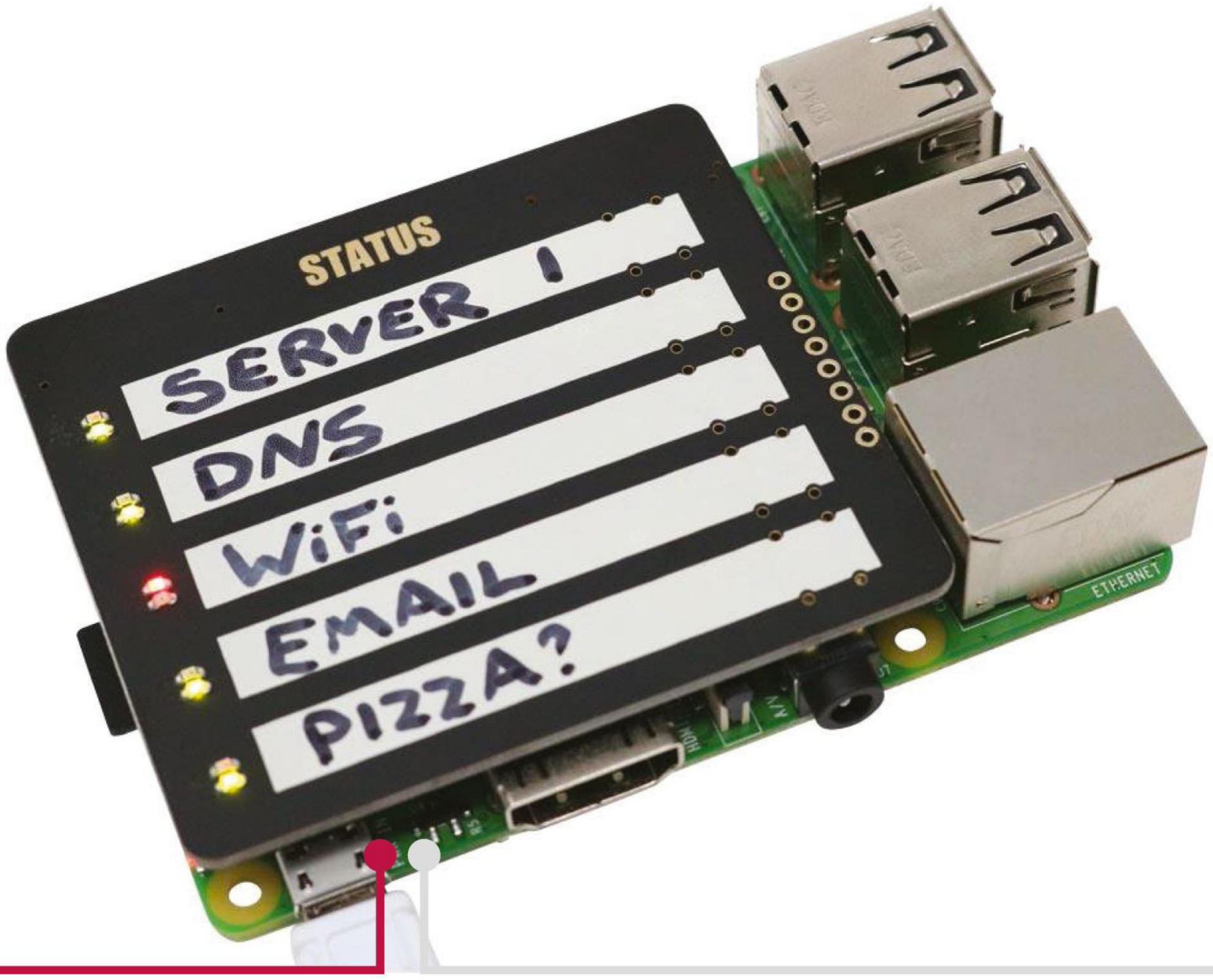


Python and Minecraft Pi

Using Python, we can hook directly into Minecraft Pi to perform complex calculations, alter the location of our player character and spawn blocks. We can do pretty much anything from creating prefabricated pixel art, to communicating directly with the player via in-game chat. Now, with this issue's tutorial, we can do all of it over the network. By hacking into our friend's Minecraft Pi, we can manipulate their game world and their player character to our heart's content.

Keep track with Status Board

Monitor your internet connection, detect who's home and keep track of your finances with this handy Pi add-on



The Pi Hut's latest add-on, the Status Board is simple yet powerful, like the Pi itself and with a board, you can monitor anything you want at a glance.

The Status Board has five customisable strips, each with a green and red LED. The options are almost limitless. Do you want to be able to quickly check if your home server is online? Or perhaps you'd prefer to monitor a value, such as whether your investments have gone up or down? Each strip can be written on with a dry-wipe marker, so you can keep reassigning them.

In this guide, we'll focus on basic setup of your Status Board, along with some example scripts coded by us and the folks at the Pi Hut. These sample scripts can be used to check your internet connection is working, ping devices, as well as check if your shares have gone up or down in value. You'll also discover how to customise the scripts to your needs and how to design your own.

This tutorial assumes you have a clean install of Raspbian Stretch on your Raspberry Pi. Make sure to run sudo apt-get update and sudo apt-get upgrade before proceeding.

01 Connect your Pi Status Board

Head over to the Pi Hut and purchase the Status Board for either your Raspberry Pi 3 or Pi Zero (**http://bit.ly/SBoardbuy**). Setup is incredibly simple as you only need to connect the Status Board directly to the GPIO pins on the Pi. Disconnect the Pi's power cable while connecting the board.

In order to use the Status Board you'll need the latest version of the GPIO Zero Python library. This should be pre-installed in Raspbian but if not, open a Terminal window and run sudo apt install **python3-gpiozero**

python-gpiozero -y.

02 Test Status Board

Open a Terminal on your Pi and launch the Python shell by typing python3. First, import the StatusBoard module by entering from gpiozero import StatusBoard. Next, enter sb = StatusBoard(). This will make it easier to type commands, as they'll be shorter.

Next, switch on all the LEDs on all strips by running sb.on(). Switch them off again by typing sb.off().

Now let's take a look at how to turn on a single LED. For instance, to illuminate the green LED on strip three, type: sb.three.lights.green.on(). Next, try making a light blink: sb.five.lights.blink().

```
File  Edit  Tabs  Help
pi@raspberrypi:~ $ python3
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from gpiozero import StatusBoard
>>> sb = StatusBoard()
>>> sb.on()
>>> sb.off()
>>> sb.three.lights.green.on()
>>> sb.five.lights.blink()
>>> sb.off()
>>>
```

03 Test naming strips

By default, the strips on the Status Board are named 'one', 'two', 'three' and so on. To avoid confusion, you can name strips individually in Python. Do this by defining the StatusBoard values; for instance if you're using it to detect motion in rooms in your house, you could enter: sb = StatusBoard('kitchen', 'lounge', 'basement', 'study'). If you're composing a script, you could switch on the red LED for the 'study' strip with the command: sb.study.

lights.red.on().

Type exit() once you've finished testing.

04 Download ping script

In a Terminal, run:

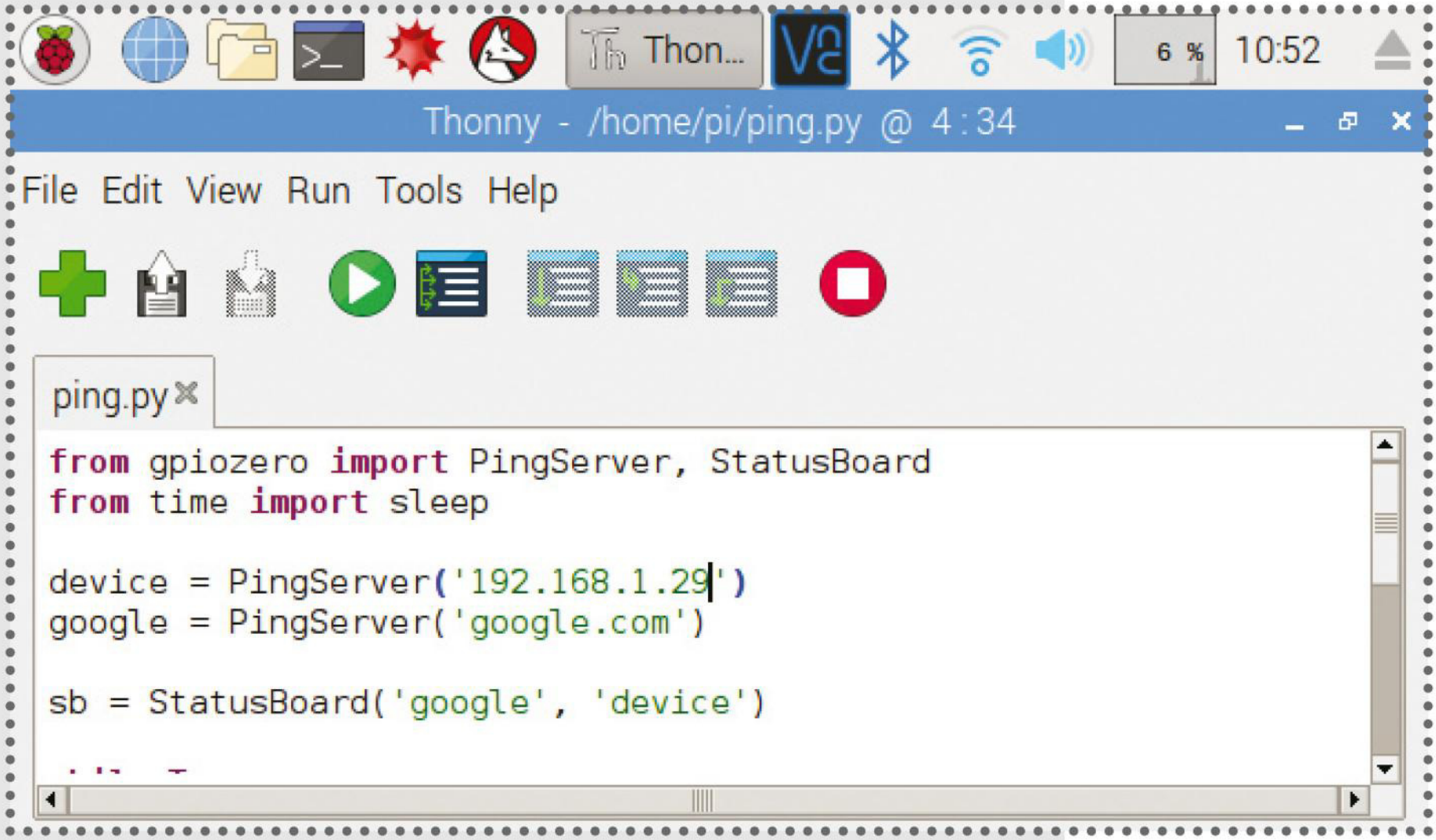
```
wget https://raw.githubusercontent.com/nate-drake/pistatusboardsamples/master/ping.py
```

This example ping script uses the first two strips on the Status Board. The first will show a blinking green light if the Pi can connect to the internet. The second checks if the IP address of a certain device such as a mobile phone is connected to your local network. To start editing, run nano ping.py in the Terminal or better yet, open it with Thonny Python IDE to ensure text is formatted correctly. Note that the first two strips have been named 'google' and 'device'.

05 Configure domain settings

Find the line which which reads google = PingServer('google.com'). Under the current

```
pi@raspberrypi:~ $ wget https://raw.githubusercontent.com/nate-drake/pistatusboardsamples/master/ping.py
--2017-09-24 10:49:20--  https://raw.githubusercontent.com/nate-drake/pistatusboardsamples/master/ping.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.120.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.120.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 494 [text/plain]
Saving to: 'ping.py'

ping.py             100%[===================>]     494  --.-KB/s    in 0s

2017-09-24 10:49:20 (2.93 MB/s) - 'ping.py' saved [494/494]

pi@raspberrypi:~ $
```



configuration, the PingServer module will check if it can connect to http://google.com every three seconds. (You can change this by amending sleep (3).)

If the connection is successful, the script will cause the green LED on the first strip to start blinking, otherwise it will switch off. The advantage of using a domain name here is that PingServer will not only check that your internet connection is working but that your DNS settings are correct too.

06 Configure device settings

The ping.py script also tries to locate a device on the same network such as a family member's mobile. To use this feature, you need to know the device's IP (the cross-platform, free mobile app Fing is perfect for this.)

Find the line which reads device =

PingServer('192.168.1.29') and amend 192.168.1.29 to the IP address of the device you want to detect. If it can be pinged, the green LED on strip two lights up, otherwise the red LED will show. Run the script from Terminal with python3 ping.py.

07 Download finances script

Reopen Terminal and run:

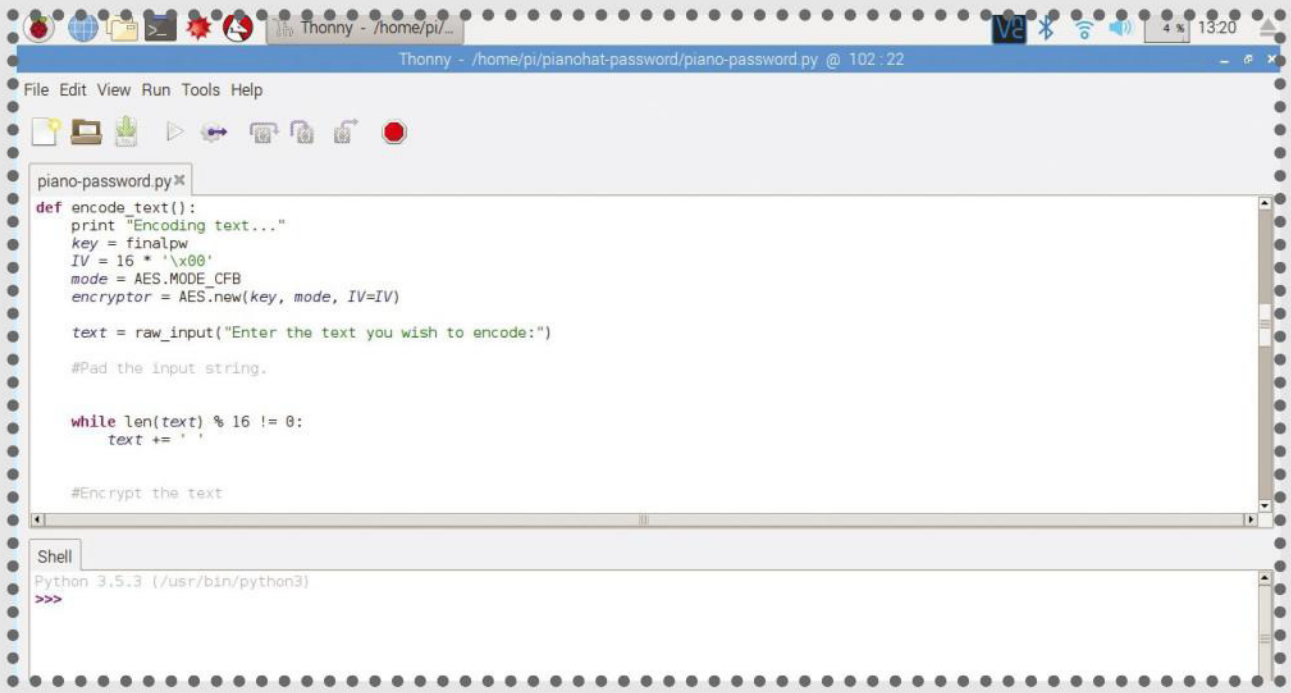
```
  wget https://raw.githubusercontent.com/nate-drake/pistatusboardsamples/master/finance.py
```

This second Python example script focuses on the third and fourth strips on the Status Board, and uses the yahoo_finance module to monitor your financial data. The first part of the script checks whether the value of your shares have gone up or down. The second part checks the current exchange rate for a currency pair. To start editing, run nano ping.py in Terminal or better yet, open it with Thonny Python IDE to ensure the text is formatted correctly.

08 Configure share price settings

If you dabble in the stock market, you'll know that shares are identified by a ticker symbol. If you find the line which reads share = Share('AAPL'), you'll see that by default the script will check the value of Apple shares. Amend this to the ticker symbol for the shares you hold. If you're not sure what this is, visit https://finance.yahoo.com to search for a company.

Make sure to amend the line share_price = 150.53 to the actual dollar value you paid per share originally too.



09 Configure currency settings

The script can also check the value of a currency pair. Find the line reading currency_pair = Currency('BTCUSD'). By default the script checks the current value of Bitcoin in USD; however, you can change this to another pair if you wish, e.g. EURPLN.

Whichever currency you choose, make sure to amend the line currency_price = 4087.45 to reflect what you paid for the base currency; e.g. you may have paid 4.2684 euros per Polish zloty. Run the script with python3 finance.py. It can run simultaneously with ping.py.

10 Intense pulses

The Status Board can control the intensity of LEDs. You can do this firstly by adding pwm=True to sb in your scripts; for instance:

```
sb = atusBoard('google', 'device', 'shares',
'xr', 'weather', pwm=True).
```

You will now be able to control the intensity of LEDs; for

instance,

```
sb.weather.lights.green.value = 0.5
```

instructs the Status Board to lower the green LED on strip five to half its usual brightness, then switch it on.
You can also use this feature to make LEDs pulse. To do this for the same LED above, run: sb.weather.lights.green.pulse(). You can also make both LEDS pulse by running:

```
sb.weather.lights.pulse().
```

Switch off LEDs in exactly the same way as we've outlined above.

11 Further steps

We have intentionally not addressed the fifth strip so that you can use it for whatever purpose has popped into your head while reading this. Head over to http://bit.ly/SBoardexamples for a few code examples. These include a script to check the London tube lines, one for checking the weather and even a program to check for names in the news.
Make sure you name your fifth strip in the way outlined above to reflect your chosen purpose. If any rogue LEDs switch on, go to **Step 2** and repeat the commands up to sb.off() to turn them off.

On-board object recognition

We look at how to do on-board image processing using OpenCV, and show you how to use it to code a self-tracking turret

This issue, we'll look at using OpenCV to add on-board image processing and object recognition to your project. To install OpenCV, use the following command:

```
sudo apt-get install python-opencv
```

This will install the Python wrappers for OpenCV. You will, of course, also need a camera to capture the imagery for processing. While you can use a USB camera, for this tutorial we'll assume you have the official Pi Camera Module attached. To use it, enter sudo raspi-config, select '5 Interfacing Options', then 'P1 Camera' and select <Yes> to enable the camera.

The first step is to actually collect an image. With the Pi Camera Module, you'll need to import several classes to handle this part:

```
from picamera.array import PiRGBArray
from picamera import PiCamera
camera = PiCamera()
camera.resolution = (160,120)
```

```
  camera.framerate = 32
  rawImgCapture = PiRGBArray(camera,
size=(160,120))
```

Once you have an image loaded, you can start image processing and trying to identify objects. Traditional algorithms would try to analyse each section of an image, testing for each of the parameters that make up the object you're trying to identify. This approach quickly overwhelms any processing power you have available.

A new technique is to use something called a cascade to try to minimise the amount of computing that needs to be done before your code can make an approximate decision about whether an object probably exists within an image.

The basic idea is that object identification is broken into stages, going from a coarse match to finer and finer detailed matches. The algorithm only cascades to the more detailed matching stages if the coarse ones pass first. This way, it doesn't waste time on regions of the image that probably don't contain the object you are trying to identify.

These cascades, within OpenCV, are just XML files that contain data that OpenCV uses to identify the object in question. You can find these cascade files at several locations online. The core cascades are available from the main repository of OpenCV. Get them with:

```
git clone https://github.com/opencv/opencv.git
```

You can then find the cascades in opencv/

data/haarcascades. With these cascades, you can create classifiers that can be used to do the image recognition. The following code handles the initialisation steps.

```
  import cv2
  cascadepath = '/path/to/cascade.xml'
  currcascade = cv2.
CascadeClassifier(cascadepath)
```

To simplify the maths involved, you need to convert the image to greyscale. The OpenCV Python module includes a number of helper functions that can manage tasks like this. This code will do the conversion:

```
  grayimage = cv2.cvtColor(rawImgCapture, cv2.
COLOR_BGR2GRAY)
```

Now, you can finally do some object recognition. Let's say you're trying to recognise faces. The following code will give you a list of all of the faces which exist within the given image:

```
  currfaces = currcascade.detectMultiScale(
    grayimage, scaleFactor=1.1, minNeighbors=5,
       minSize=(30,30), flags=cv2.cv.HAAR_SCALE_
IMAGE)
```

The first parameter is the image data itself. Next is a scale factor, which tries to compensate for objects that may be closer or further away from the camera. The minNeighbors parameter specifies how many

Why Python?

It's the official language of the Raspberry Pi. Read the docs at python.org/doc

nearby items are detected before the algorithm decides that the searched for object has been found. The minSize parameter defines the window size used for the actual object detection within the image. All of these parameters can be adjusted to best fit the image quality, along with the type of object that you wish to detect. Each of the returned sets of values in the list currfaces contains the locations of bounding rectangles surrounding each of the detected objects. These values are given as an x and y set defining the start of the bounding rectangle, along with its width and height.

This code works great for the Pi Camera Module, but the OpenCV Python module allows you to talk to USB-connected cameras and do continuous image capture and analysis. Let's say that we only have a single camera attached. We can read images with the following code:

```
video_camera = cv2.VideoCapture(0)
return_status, curr_image = video_camera.
read()
```

The first line connects to the first available USB camera. The second line reads the next available frame from the given video stream. It not only returns the image data, but also returns a status code. This is used to tell whether we are out of image frames, which can happen if we're reading from a video file. Since we're using a video camera, this won't happen. The returned image frame can then be processed in the same way as the earlier steps. Once you are done monitoring your video stream, you need to clean up.

The following code makes sure that everything gets shut down properly:

```
video_camera.release()
```

All of this is great for identifying objects within an image or video stream, but what if you need to monitor this object as it moves across the area of interest? As an example you could use this to code a self-tracking Nerf gun to discourage trespassers. To do this, we need to start with an initial location within the image. It could either be the location found using the earlier code, or you may wish to use a hard-coded set of values. The first step is to create a region of interest and the histogram so that we can track the object later:

```
roi = curr_image[r:r+h, c:c+w]
hsv_roi =  cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
  mask = cv2.inRange(hsv_roi, np.array((0.,
60.,32.)), np.array((180.,255.,255.)))
  roi_hist = cv2.calcHist([hsv_
roi],[0],mask,[180],[0,180])
  cv2.normalize(roi_hist,roi_hist,0,255,cv2.NORM_
MINMAX)
```

The initial location of interest is given by the values of r, c, w and h. As we're using NumPy arrays in the third line of code, you'll need to add the following line to the top of the program:

```
import numpy as np
```

You also need to set some kind of termination

"Object identification is broken into stages"

condition so that it stops at some point. For example, the following code sets this condition to be either 10 iterations or movement of at least 1 point:

```
  term_criteria = (cv2.TERM_CRITERIA_EPS | cv2.
TERM_CRITERIA_COUNT, 10, 1)
```

Now that everything is initialised, you can start looping through the image frames in your video stream and tracking your object's movement with the following code:

```
  ret, curr_image = video_camera.read()
  hsv = cv2.cvtColor(curr_image, cv2.COLOR_
BGR2HSV)
  dst = cv2.calcBackProject([hsv],[0],roi_
hist,[0,180],1)
  ret, track_window = cv2.meanShift(dst, track_
window, term_crit)
```

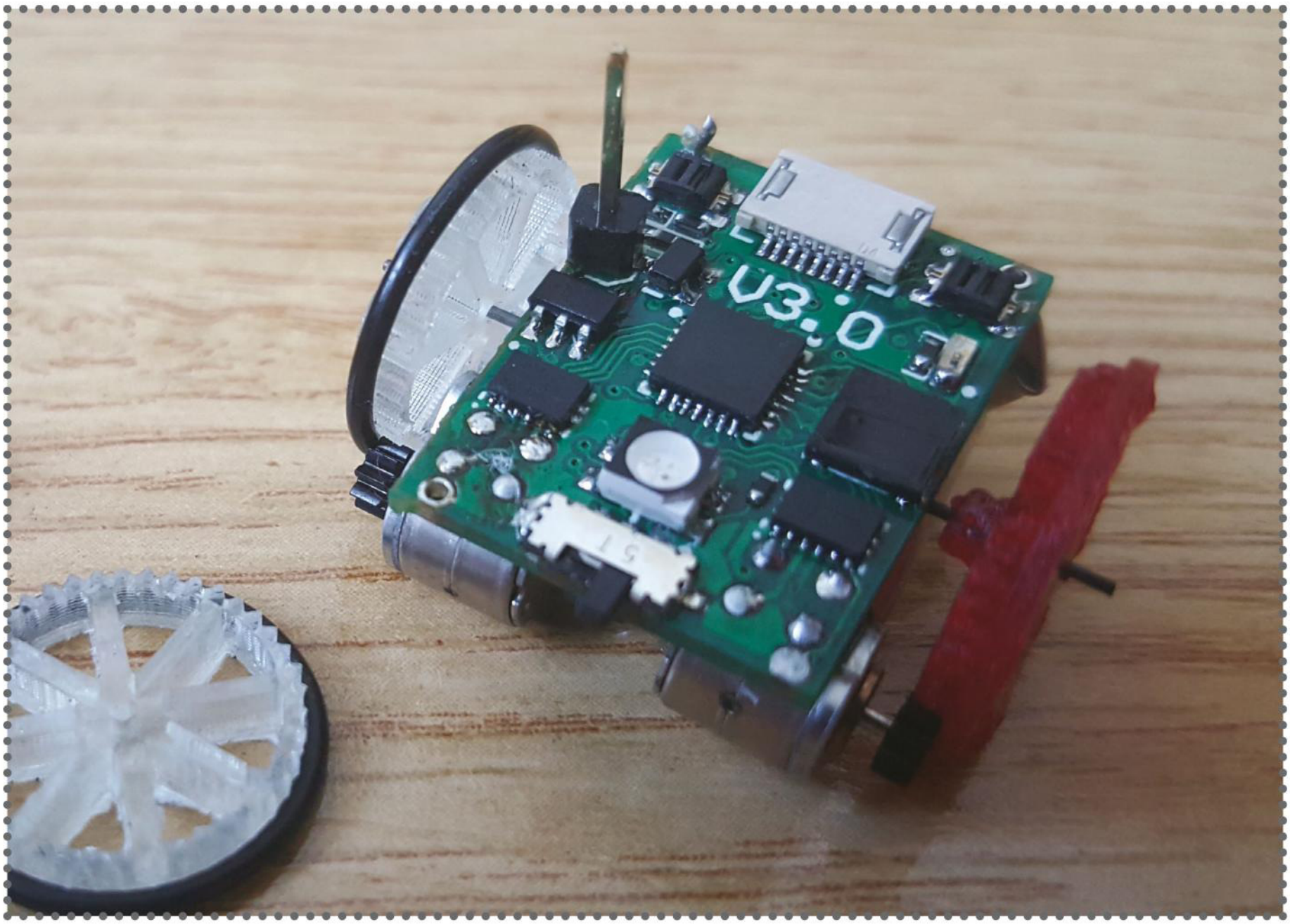
The first line captures the next frame. The second line converts it to hues so that it can be compared to the histogram generated above. The third line does the back projection to figure out whether the object has moved, and the fourth line gives the new region of interest for the current image frame.

With this code, you should be able to have your project identify a given object, and then track it as it moves around. There is no reason your turret-mounted Nerf gun shouldn't be able to defend your workshop now.

Next issue

Get inspired **Expert advice** **Easy-to-follow guides**

"Attack of the micro-robots!"



Get this issue's source code at:

www.linuxuser.co.uk/raspicode